

**ROUTING AND ACTION**

**MEMORANDUM**

---

ROUTING

---

TO: (1) Mathematical Sciences Division (Pasour, Virginia)

Report is available for review

(2) Proposal Files    Proposal No.:        56757MA.4

---

DESCRIPTION OF MATERIAL

---

CONTRACT OR GRANT NUMBER:        W911NF-09-1-0538

INSTITUTION:    Virginia Polytechnic Institute & State University

PRINCIPAL INVESTIGATOR: Reinhard Launbenbacher

TYPE REPORT: Book

DATE RECEIVED: 9/28/2012    7:38:18AM

PERIOD COVERED:    through

TITLE: Agent-based models and optimal control in biology: a discrete approach.

---

ACTION TAKEN BY DIVISION

---

(x) Report has been reviewed for technical sufficiency and IS ☒ IS NOT ☐ satisfactory.

(x) Material has been given an OPSEC review and it has been determined to be non sensitive and, except for manuscripts and progress reports, suitable for public release.

Approved by SSL\VIRGINIA.PASOUR on 10/18/2012    11:19:48AM

ARO FORM 36-E

REPORT DOCUMENTATION PAGE				Form Approved OMB NO. 0704-0188	
<p>The public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington VA, 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to any penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number.</p> <p>PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.</p>					
1. REPORT DATE (DD-MM-YYYY) 28-09-2012		2. REPORT TYPE Book		3. DATES COVERED (From - To) -	
4. TITLE AND SUBTITLE Agent-based models and optimal control in biology: a discrete approach.				5a. CONTRACT NUMBER W911NF-09-1-0538	
				5b. GRANT NUMBER	
				5c. PROGRAM ELEMENT NUMBER 611102	
6. AUTHORS Franziska Hinkelmann, Matt Oremland, Reinhard Laubenbacher				5d. PROJECT NUMBER	
				5e. TASK NUMBER	
				5f. WORK UNIT NUMBER	
7. PERFORMING ORGANIZATION NAMES AND ADDRESSES Virginia Polytechnic Institute & State University Office of Sponsored Programs Virginia Polytechnic Institute and State University Blacksburg, VA 24060 -				8. PERFORMING ORGANIZATION REPORT NUMBER	
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) U.S. Army Research Office P.O. Box 12211 Research Triangle Park, NC 27709-2211				10. SPONSOR/MONITOR'S ACRONYM(S) ARO	
				11. SPONSOR/MONITOR'S REPORT NUMBER(S) 56757-MA.4	
12. DISTRIBUTION AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.					
13. SUPPLEMENTARY NOTES The views, opinions and/or findings contained in this report are those of the author(s) and should not be construed as an official Department of the Army position, policy or decision, unless so designated by other documentation.					
14. ABSTRACT This chapter describes an approach to the optimal control of complex biological systems. Such systems can often be modeled effectively through agent-based computational models. To apply rigorous mathematical methods to synthesize optimal control strategies one approach is to approximate the agent-based model through a mathematical model for which analytic methods are available. One such framework is that of time-discrete, state-discrete dynamical systems, which can be described as polynomial dynamical systems over a finite field. The chapter					
15. SUBJECT TERMS agent-based model, optimal control, biological network, discrete model					
16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT UU	15. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON Reinhard Laubenbacher
a. REPORT UU	b. ABSTRACT UU	c. THIS PAGE UU			19b. TELEPHONE NUMBER 540-231-2100

## **Report Title**

Agent-based models and optimal control in biology: a discrete approach.

### **ABSTRACT**

This chapter describes an approach to the optimal control of complex biological systems. Such systems can often be modeled effectively through agent-based computational models. To apply rigorous mathematical methods to synthesize optimal control strategies one approach is to approximate the agent-based model through a mathematical model for which analytic methods are available. One such framework is that of time-discrete, state-discrete dynamical systems, which can be described as polynomial dynamical systems over a finite field. The chapter presents algorithms, examples, and associated software for these tasks.

# **Agent-based models and optimal control in biology: a discrete approach**

Reinhard Laubenbacher, Franziska Hinkelmann, Matt

Oremland

Virginia Bioinformatics Institute, Virginia Tech

[reinhard@vbi.vt.edu](mailto:reinhard@vbi.vt.edu)



## Chapter 12

Agent-based models and optimal control in biology: a discrete approach

### NON-PRINT ITEMS

**Abstract:** Agent-based modeling is an increasingly important tool in computational biology, since it allows the construction of mathematical models for systems for which it is difficult or impossible to give a global description through, e.g., a system of differential equations. Agent-based models are particularly well-suited for capturing phenomena of “emergent” global dynamics from local interaction rules. Balancing these features are some drawbacks. An important one is the difficulty of analyzing such models through mathematical tools, so that simulation is often the only available methodology. This shortcoming represents a particular hindrance when it comes to the use of such models for the purpose of optimal control, frequently an objective in ecology and in biomedicine, for instance. Without rigorous mathematical methods, the discovery of optimal control strategies becomes a purely heuristic process relying solely on simulation.

This chapter will outline an approach to providing some mathematical analysis tools for agent-based models by translating them into the framework of polynomial dynamical systems over a finite field. Through a faithful translation, what is essentially a computer program now becomes a rigorously described dynamical system within the framework of computational algebraic geometry which provides many theoretical and computational tools for finding the long-term dynamic behavior of an agent-based model and for finding and implementing optimal control strategies.

**Keywords** Agent-based model, global dynamics, polynomial dynamical system, algebraic framework, discrete mathematics, optimal control, optimization, heuristic

## 1. Introduction

The need to control complex systems, both engineered and natural, pervades our lives. From the thermostat that controls the temperature in our homes to the software that controls flight characteristics of the space shuttle during landing, the vast majority of engineered systems have built-in control mechanisms. Being able to control certain biological systems is no less important. For instance, we need to control ecosystems for agriculture and wildlife management. We need to control different parts of the human body to cure diseases such as hypertension, cancer, or heart disease. And we need to control microbes for the efficient production of a vast array of biomaterials. With the need for control comes the need to carry it out in a fashion that is optimal with respect to a given objective. For instance, we need to devise a schedule for administering radiation therapy to cancer patients in a way that maximizes the number of cancer cells killed while minimizing side effects. We need to apply pesticides to fields in a way that minimizes environmental damage. And we need to control the metabolism of engineered strains of microbes so they produce the maximal amount of biofuel. Thus, the need for *optimal control* is a problem we face everywhere. This chapter will focus on optimal control of biological systems.

The most common approach to optimal control is through the use of mathematical models, often consisting of one or more (ordinary or partial) differential equations. These equations model key features of the system to be controlled and include one or more variables that represent control inputs. The following example illustrates this approach, taken from [21], where more details can be found. The focus is on the optimization of cancer chemotherapy, taking into account certain immunological activity. The two relevant variables are  $x$ , which represents the volume of the tumor to be treated, and  $y$ , the immunocompetent cell density capturing various types of T-cells activated during the immune reaction to the tumor. These

two variables are governed by the system of ordinary differential equations

$$\begin{aligned} dx/dt &= \mu_C F(x) - \gamma xy, \\ dy/dt &= \mu_I(x - \beta x^2)y - \delta y + \alpha. \end{aligned}$$

Here,  $F$  is a function that represents the carrying capacity of the tumor, a concept adapted from ecology, which represents that ability of the environment to sustain the tumor. The Greek letters are constant parameters of the models. For instance,  $\delta$  represents the rate of natural death of T-cells. For our purposes the exact model structure is not important, and details can be found in [21]. For instance, the term  $-\beta x^2$  represents the observation that tumors suppress the activity of the immune system.

The control objective here is to reduce the tumor volume through the use of a chemotherapeutic agent. We implement this control with a term  $-xu$ , where  $u = u(t)$  represents the control input, namely the amount of the chemotherapeutic agent administered. The factor  $x$  takes account of the assumption that the effect of chemotherapy on the tumor is proportional to the tumor volume. Thus, the first equation becomes

$$dx/dt = \mu_C F(x) - \gamma xy - \kappa xu.$$

What kind of drug regime is appropriate for a given cancer patient depends in part on the particular state of this patient's disease. Our goal now is to optimize this treatment with respect to several factors. On the one hand we want to shrink the tumor through the administration of a maximal dose of the agent, and on the other hand we need to take into account the toxic side effects of the medication. Also, the treatment should be as short as possible. All this can be combined into the cost function

$$J(u) = ax(T) - by(T) + \int_0^T (cu(t) + d)dt.$$

This equation represents the cost of applying treatment schedule  $u$  for a length of time  $T$ . The optimal control problem now is to find a control  $u$  that minimizes this cost function. Solving

such control problems is an active area of research, and algorithms have been developed to find  $u$  [24]. See also [13] for optimal control problems in systems biology.

Modeling biological systems through differential equations has its limitations, however. In many cases, the processes involved might be fundamentally discrete rather than continuous. For instance, in the case of a predator-prey relationship between two species inside an ecosystem, both populations are comprised of discrete individuals that engage in typically binary discrete interactions. Thus, it is not immediately clear whether one can apply methods such as differential equations, which assume that the quantities modeled vary continuously. In molecular biology, when we study regulatory relationships between genes inside a cell, these relationships are based on the interactions of discrete molecules. Modeling such systems using differential equations is based on two assumptions: first, there are many individuals involved, so that we can view them collectively as a continuous quantity; second, that we are able to describe the individual interactions in a “global” manner, as a term in a differential equations, usually involving one or more global parameters. Sometimes, both of these assumptions are justified, such as for large populations of bacteria or large quantities of chemicals in a fermenter. But at other times, one or the other, or both, of these assumptions fail. In a cell, there might only be 2 or 3 molecules of a particular protein present at any given time, so that its role in a regulatory network becomes discrete and stochastic, and cannot be accurately modeled with continuous functions. And in an ecosystem with several species, continuous models sometimes cannot accurately capture extinction events, when one species reaches a very low count.

For these reasons, and others, several other frameworks have been used to model biological systems. One of these relies on so-called “agent-based” or “individual-based” models. This type of model has a long history, going back to the 1940s and the work of John von Neumann [27].

Von Neumann was interested in the process of self-replication and aimed to construct a machine that could faithfully replicate itself. A theoretical instantiation of such a machine

turned into the concept of a *cellular automaton* as a computational model. A very well-known example of a cellular automaton is John Conway's *Game of Life* [4]. Since it includes many of the basic concepts of agent-based models, we describe it briefly.

The *Game of Life* takes place on a chess-board-like 2-dimensional grid. This grid can either be finite or extend infinitely in all directions, thereby obtaining two different versions of the game. Each square, or *cell*, on the grid can be either black or white. Since the *Game of Life* is intended to simulate life, a cell is instead referred to as either alive (black) or dead (white). Each cell away from the boundary of the grid has 8 neighbors on the grid that physically touch it, 4 with which it shares an edge, and 4 that touch only on the corners. Cells on the boundary have fewer neighbors. To make all cells uniform, one can make the assumption that a cell at an edge of the grid, but away from the corner, has as additional neighbors the corresponding cells on the opposite edge of the grid. Thus, we effectively “glue” opposite edges together, so that the grid is situated on a torus rather than in the plane. Now that all cells have 8 neighbors we establish a rule that determines the evolution of the cells on the grid by determining what the status of a given cell is, depending on the status of its 8 neighbors. The rule is quite simple.

- Any live cell with fewer than 2 live neighbors dies;
- Any live cell with 2 or 3 live neighbors stays alive;
- Any live cell with more than 3 live neighbors dies;
- Any dead cell with 3 live neighbors comes alive.

Thus, the rules are reminiscent of a population whose survival is affected by under- and overpopulation. If we now initialize this “Game” by assigning a “live” or “dead” status to each of the cells, then we can use this rule to evolve life on this grid by updating the status of all the cells in discrete time steps. The result is a vast array of different dynamics that can be observed, which is largely unpredictable. There is a rich literature on this topic and many websites with *Game of Life* simulators.

General agent-based models have many features similar to this set-up. There is a collection of agents that are distributed across a spatial landscape. In the *Game of Life*, the spatial landscape is the grid, and there is one agent per cell. Each agent can be in one of a (typically finite) number of states, such as “dead” or “alive,” and has a set of rules attached to it that it uses to determine its state based on the state of those other agents it interacts with at any given time. In general, agents are able to move around in the spatial landscape, and there are rules that determine the agents’ movement patterns. Typically, the rules are stochastic rather than deterministic and are governed by a collection of probabilities. For instance, agents might be predisposed to follow a certain gradient, representing, e.g., nutrient availability, but there is some chance that they might move in a different direction. While there are many other variations and features in particular agent-based models, these few basic features characterize the agent-based modeling framework. They also are sufficient to explain the basic differences between agent-based models and equation-based models, such as ordinary differential equations.

Agent-based models lend themselves very well to a description of dynamical systems that arise from local interactions of many parts/agents, based on local rules only rather than on the configuration of the entire system at any given time. And it is very easy to represent a rich heterogeneous spatial environment that the agents navigate. Thus, the dynamics of the entire system, or its so-called global dynamics, “emerges” from these local interactions by applying the local rules repeatedly. In contrast, a system of differential equations, for instance, explicitly describes the global dynamics of the system up front. Furthermore, all the specifications for an agent-based model are intuitive, in the sense that they are direct computational representations of recognizable features in the actual system. This leads to models that are more faithful to the system to be modeled and that are more accessible to domain experts. With existing software for model building, they can even be built by domain experts directly, without the intervention of a modeler or mathematician. But, as always, there are no free lunches. These advantages come with some significant costs attached.

The reason that agent-based modeling became widespread only in the last 15 years or so is due to the fact that larger models with more features require extensive computational resources that were not available until recently. It is only now possible, barely, to build and analyze models that might have hundreds of millions of agents and tens of millions of spatial locations, with agents being very complex in terms of the states they can take on and the rules they follow. Even moderately complex models require high performance computing facilities for their analysis, which makes it difficult for individual researchers to use them. High computational complexity is compounded by the fact that there are very few mathematical tools available for the analysis of agent-based models. In essence, the only approach to model analysis is model simulation. That is, from given initializations of agent states and environmental parameters, one observes the time evolution of the system to draw conclusions about, e.g., steady states of the model. Through choosing many initializations and doing many simulation runs from a given initialization in the case of a stochastic model, one aims to obtain global dynamic information about the model. Little else can be done because, in essence, the model consists of a computer program rather than a set of mathematical equations, and there are few things one can do with a computer program other than run it.

The lack of mathematical analysis tools extends in particular to optimal control. Existing approaches are heuristic in nature, based on domain knowledge. The goal of this chapter is to describe some of these approaches and to outline steps one can take to expand the repertoire of available tools to include techniques based on mathematics. The way to do this is to translate an agent-based model into a mathematical object, such as a system of equations of some sort, that makes it amenable to mathematical analysis tools. There are several possible ways in which to do this, and research is only in its early stages. Thus, the reader should see this chapter as a snapshot of an exciting research area that is evolving rapidly and provides rich opportunities for contributions at all levels. This chapter showcases one possible approach and the steps that have been accomplished on the road to developing mathematical tools for optimal control of agent-based models.

## 2. A First Example

Go to <http://ccl.northwestern.edu/netlogo/models/RabbitsGrassWeeds>. There you will find an agent-based model of rabbits in a field of grass and weeds. At each time step (or ‘tick’) the rabbits move in a random direction (they lose energy by moving). If there is grass at their location, they eat it and gain energy. If their energy level climbs above a certain threshold, they give birth (in this model, a new rabbit is spontaneously created at the location of the parent). Upon birth, the parents’ energy is halved, and the offspring is created with this halved energy level. Upon each tick, empty squares (or ‘patches’) have a certain fixed probability of grass re-generating. Weeds can also be introduced in order to further complicate the dynamics; their behavior is similar to that of the grass.

Near the top of the page you will find an option that allows you to run this model in your web browser. Spend some time reading through the description in order to get a feel for how this model works. Click **setup** and then **go** to run the simulation (press **go** again to stop the simulation). Note that you can speed up or slow down the model by using the slider at the top of the interface. Each time you wish to re-initialize the model and start over, you must click **setup** again.

In this model, each of the grid squares (henceforth referred to as *patches*) are agents, and each rabbit is an agent as well. Notice that the rabbits move around randomly, eating grass as they encounter it. Note too that the patches are colored green when grass is present, and black when it is absent. Notice the sliders on the left-hand side: these values can be changed as the simulation proceeds, or prior to starting a simulation. What do you expect to happen to the rabbit population if the grass growth is set to a very low rate? What if the grass grows quickly? What effect would altering the birth threshold have on the amount of grass present at each time step? Do you think it is possible for the rabbit population and grass level to stabilize over time? Play around with the sliders to determine if your predictions are correct. Note that you may need to let the simulation run for several hundred time steps

(or ‘ticks’ in Netlogo) in order to observe consistent dynamics. Do the populations stabilize? Do they oscillate?

### 3. Netlogo: an introduction

Hopefully, the example in Section 2 has given you an idea of what an agent-based model is: a computer simulation wherein agents interact with their local environment (possibly including other agents) based on simple rules. In this section we guide you through several exercises involving agent-based models using Netlogo [28], a software tool developed to work with agent-based models. Netlogo can be downloaded for free at <http://ccl.northwestern.edu/netlogo/>. There you will find detailed instructions on how to install it. Netlogo is its own programming language, so named because it is a variation of the Logo language. While many key features are unique to Netlogo, users familiar with Logo will likely note similarities. Note too that Netlogo is continually updated and newer versions released. As of the time of this writing, the latest version is Netlogo 5.0. All files and exercises associated with this chapter will be conducted using this version; some adjustment may be necessary for newer (or older) versions. While there are many other software platforms available and in use for agent-based modeling, for the remainder of this chapter we will use Netlogo to introduce and examine agent-based models. It has the convenient advantage of providing the user with an intuitive graphical interface, which we will use to aid our understanding of the models we will examine. In addition, the standard installation comes with a library of models, all of which are open-source. Thus we may build models from scratch or we may choose to alter an existing model. You may wish to go through the tutorials found at <http://ccl.northwestern.edu/netlogo/docs/>. A complete dictionary of programming terms can be found at <http://ccl.northwestern.edu/netlogo/docs/dictionary.html>. All exercises in this section refer to the “Rabbits Grass Weeds” model introduced in Section 2. It should be noted that for Exercises 12.1 and 12.2, the web-based version of the model can be used (via <http://ccl.northwestern.edu/netlogo/models/RabbitsGrassWeeds>), but Exercises 12.3 and 12.4 require installation of the Netlogo software.

EXERCISE 12.1. Note the two sliders **weeds-grow-rate** and **weed-energy**. By setting **weeds-grow-rate** to be any non-zero value, you will notice that the landscape of the model has been altered, as there are now patches containing weeds interspersed with the grass (represented as purple patches). What effect does this have on the rabbit population? How about on the grass population? What happens when you increase **weed-energy**? Set **grass-grow-rate** and **weed-grow-rate** to be equal, and set **grass-energy** and **weed-energy** to be equal as well. What do you think will happen to the population levels now? Determine slider values so that the following situations occur:

- (a) The grass and weeds stay at (roughly) the same level (as shown in the plot window).
- (b) The weed levels (as shown in the plot window) are higher than the rabbit levels and the rabbits die out.
- (c) The weed levels (as shown in the plot window) are higher than the rabbit levels and the rabbits do not die out.

EXERCISE 12.2. Set **weed-grow-rate** and **weed-energy-level** to 0, and **grass-grow-rate** and **grass-energy** to 5 if they are not already at those levels. Set **birth-threshold** to 15, and press **setup** and **go** to restart the simulation. As the simulation runs, gradually decrease **birth-threshold** to 10, and then to 5. You will notice in the plot window that the rabbit population oscillates at a higher amplitude as you decrease **birth-threshold**. Why does this happen? As you further decrease **birth-threshold** to 2 or 1, the rabbits die out. Why does this happen?

EXERCISE 12.3. Right-click on the graphical interface and select **Edit...**. Notice that the boxes labeled ‘**World wraps horizontally**’ and ‘**World wraps vertically**’ are checked. This means that when a rabbit moves left from a left-most patch, it will reappear on the corresponding right-most patch (and similarly if the rabbit moves vertically from a top-most patch). If you uncheck those boxes, the rabbits will be bound by the edges of the map (so we can think of them as being ‘fenced in’). What effect does checking these boxes have on the rabbit population? What effect does it have on the grass?

EXERCISE 12.4. By altering the code in the ‘Code’ tab, change the patches in the model so that there is a river that separates the field into two halves. Further alter the code so that grass and weeds cannot grow in the river, and rabbits cannot move across the river. What effect does this have on the rabbit population over time? What if weeds are introduced on one side of the river but not on the other? What happens if we do not allow the world to wrap vertically or horizontally (see Ex. 12.4)?

#### 4. An Introduction to Agent-Based Models

Mathematical modeling is a method of encoding relationships and interactions in a natural or engineered system into a formalized system; the models can then be studied and analyzed using a variety of mathematical approaches. Models allow researchers from a wide variety of disciplines to examine systems and their emergent behavior. For example, a model may be used in order to make predictions about the future behavior of a system, or it may be used to solve a complicated problem explicitly. Agent-based models are a class of computer models in which entities (referred to as agents) interact with each other and/or their local environment. Formally:

DEFINITION 12.1 (Agent-based model). A computer model that consists of a collection of agents/variables that can take on a typically finite collection of states. The state of an agent at a given point in time is determined through a collection of rules that describe the agent’s interaction with other agents. These rules may be deterministic or stochastic. The agent’s state depends on the agent’s previous state and the state of a collection of other agents with whom it interacts. [19]

By allowing agents to assume only a finite number of possible states and by placing the dependence of those states on a series of rules, agent-based models are well-suited for computer implementation. Systems such as the human immune system are increasingly being implemented in the form of agent-based models (with individual cells as the agents,

of which there may be many types) as more and more research involves the use of *in silico* simulation to study the properties of this and other similarly complex systems.

Agent-based models have the advantage of being well-suited for modeling many different types of systems. They have been used to study social interactions among individuals, the spread of disease through populations, scheduling and efficiency of factory processes, how cells react to drug treatments, and many other systems. It is perhaps worth noting that many of the current issues in the scientific community are interdisciplinary. Finding a cure for cancer will involve geneticists, biologists, mathematicians, chemists, and perhaps many other specialists. Agent-based models have an intuitive formulation and can often be examined via a graphical interface. Thus they are a natural tool for promoting interdisciplinary research, as the mathematics underlying the models is hidden in the programming; in other words, it is possible for biologists, chemists, and other researchers to make use of agent-based models without a full background in the mathematics that are involved in creating the model. At the same time, the mathematical structure remains and can be explored concurrently by mathematicians.

In developing the best drug to administer to treat a particular disease, researchers in the past have relied on trial-and-error methods via repeated experimentation on live subjects. With an accurate model of the subject, however, such experiments can often be faithfully reproduced *in silico*; that is, they can be run on a computer and analyzed with far less preparation and expense. In addition, computer simulation saves time: given an accurate model of the immune system, a year's worth of real time can be simulated in several minutes. Experimentation remains part of the process - a model will always have a limited ability to predict and must be correlated against empirical data in order to ensure that the models are indeed faithful simulations of the actual physical system. Thus, models are best used to inform *in vivo* experimentation, which in turn produces results that can be used to calibrate the model further.

Even within the scope of mathematical modeling, there are several noteworthy advantages to agent-based modeling. One such advantage is that they are effective for modeling systems wherein many agents follow the same set of rules (e.g., rabbit populations or blood cell types). In particular, models containing spatial heterogeneity can be effectively represented via agent-based models while perhaps not so easily using, for example, ordinary or partial differential equations (ODEs). Altering the spatial dynamics of an agent-based model consists of changing several lines of code or less, while such changes can be difficult (and perhaps even impossible) to implement in an ODE model.

While agent-based models provide a convenient and natural setting for studying complex systems, there are several issues within the research community that are currently unresolved. A 2006 paper [9] written by a large group of researchers identified two main obstacles: the first is the lack of standardization of the description of agent-based models, and the second is a lack of rigorous mathematical formulation of the system itself. Descriptions of agent-based models can vary in different settings, and as of this writing there is no standard definition that is universally agreed upon. Some models are developed to simulate physical processes, and others are developed in the framework of graph theory. In some cases models are developed in order to study only a certain aspect of the given system; thus the model may have more variables pertaining to certain processes than others. A research article may begin with the statement of an objective, or it may begin with a description of the model itself. In some cases, the rules that govern the updating of the agents' state variables are deterministic, and in other cases they are stochastic. All of these issues would be resolved if there was a standard protocol for describing agent-based models (indeed, one such protocol is proposed in [9]). The need for an agreed-upon structure to be followed is perhaps most clearly felt in the model's presentation. In particular, the layout of the model presentation ought to be standardized so that a reader immediately knows where to look in the description in order to learn what the model describes and what its rules are. In the current literature models are presented in myriad forms, and the descriptions of the agents, environments, and rules come

in no particular order. Thus a reader is required to scour the paper for pertinent details that might otherwise be presented in some standard way.

The second major issue concerns the lack of rigor in the formulation of the model. In many cases, the description of a model is given in several paragraphs, describing in some imprecise manner what the agents are and how they interact with each other. In fact, in order for agent-based models to be implemented on computer software, there are intricate rules embedded in the programming of the models. These rules and equations are often glossed over if not entirely omitted; if they are given, it is typically through a verbal description rather than a strict mathematical formulation. Looking back on the example in Section 2, the precise way in which the rabbits move is not described. For example, it is unclear from a simple description whether the rabbits can move diagonally, or whether the distance they move at each time step is variable or constant. In fact, even running the model does not immediately answer this question - it is only made explicitly clear by thorough examination of the code. In order to describe such a model rigorously, it is necessary that this information (and other similarly imprecise descriptions) be presented clearly and unambiguously.

In addition to these issues, an author may spend several paragraphs explaining how the model was formulated that could just have easily been given in one or two equations. Short and precise definitions can save time for the reader and also make the author's intentions explicit. One proposition to overcome this lack of formalization is presented in [10]; this work proposes a rigorous mathematical representation for agent-based models as polynomial dynamical systems. A further description of such systems can be found in [26] and is described in Section 8.

## 5. Optimization and optimal control

Agent-based models are typically created to simulate some real-world process in order to aid investigation. Once a model has been created, a natural next step is to ask: what are we to do with it? In this section we give a brief overview of optimal control and optimization, and introduce these ideas as they apply to agent-based models.

*Optimization* is the process of finding the best solution with respect to a particular goal. For example, suppose we have a model of the immune system battling a bacterial infection, and we wish to study the effects of certain drugs on this battle. We may wish to find out which drug does the least amount of tissue damage while curing the infection - this is an optimization problem, because we are searching for the best drug with respect to the stated goal. On the other hand, perhaps our goal is to cure the infection in the shortest time possible, regardless of the tissue damage caused. This would be another optimization problem. It is likely, then, that the solution to the optimization problem depends on the optimization goal. In this scenario, the drug which cures the infection the quickest may consequently do more tissue damage than other drugs (though not necessarily); on the other hand, the drug that causes the least tissue damage might not cure the infection in the shortest possible time. In this example, optimization is a process of minimization: in one case we wish to minimize tissue damage and in the other we wish to minimize the healing time. However, it is also possible that the solution of an optimization problem is a process of maximization: given a model of an immune system fighting a fatal illness, what treatment will enable the patient to survive the longest? In fact, it may be that the goal of the optimization problem is to minimize one value while maximizing another. For example, our optimization goal may be to minimize tissue damage and maximize the expected lifespan of the patient, given that we can only administer a particular drug a fixed number of times. In general, however, optimization is the process of finding the best solution, depending on the objective.

Typically, optimization is a complicated process, and becomes even more so when realistic constraints are put in place. Through the use of agent-based models it may be possible to obtain a solution to an optimization problem that is not feasible in actuality: for example, the solution may exceed monetary limitations, may require actions that are not permitted by health care regulations, or may require interaction with the patient in an impractical way (e.g., if the solution calls for treatment every hour for 100 consecutive hours, or for 100% of the population to receive vaccination). Nevertheless, optimization remains a natural means

of applying mathematics to solve real-world problems. Such problems are often framed as questions of optimization: what is the best outcome that can be produced based on the properties of the model?

*Optimal control* is a slightly different notion. In an optimal control problem, we ask the following question: given a particular state that we hope to reach, what is the most efficient way of reaching that state? In other words, we know the state of the system that we hope to reach, and the control problem is to find the solution that steers the system to that state in the best manner. Using the model of the immune system fighting an infection as an example, we may formulate an optimal control problem as follows: given that we wish to eliminate the number of infected cells in one month, what is the best drug treatment schedule we can devise? To summarize: in an optimal control problem we know the state we hope to reach and we search for the best way to reach it, whereas in an optimization problem, a goal is stated and we compare solutions to see which solution maximizes or minimizes the stated goal of the problem.

We now present an example, explaining several terms related to optimization and optimal control that will be used in Sections 6 and 7. We then conclude this section with definitions of said terms. We do this in order to standardize and clarify the terminology within this chapter. Note that while many terms are also used in optimal control for continuous systems, there might be subtle differences in their meanings when applied to agent-based models. Furthermore, as this chapter is meant to provide an introduction to the topic, lengthy formal definitions are outside the scope of this text. For a more formal treatment see [16].

**Example.** Suppose that we are modeling lung cancer and we wish to study the effect of a certain drug. We formulate our optimal control problem as follows: which treatment schedule should we choose in order to reduce the number of cancer cells so that it remains below some fixed threshold over the course of one year, given that we wish to minimize the number of times the drug is administered and maximize the number of healthy cells? (See the example given in the chapter introduction.)

In this case, there will be many variables: the number of healthy cells, the number of cancer cells, the rate at which cancer cells grow, the rate at which healthy cells regenerate, the expected lifespan of the patient, the frequency with which the drug is administered, the type of drug, and so on. Some of these variables will have fixed values: for example, the rate at which healthy cells regenerate (when no treatment is administered) can be determined through experimental measurements, and in fact this rate helps to define the model itself. Such variables are referred to as *model parameters* - they are a part of the specification of the model. The repeated interactions of the entities in the model, such as immune cells or rabbits, is referred to as the *model dynamics*. Note that we will only have direct control over some of the variables, and others we will simply measure by observation. For example, during each day of the simulated treatment we can decide whether or not to administer the drug; thus we have direct control over the value of this variable. We refer to these as *control variables*, because we have direct control over their values and they exercise control over certain aspects of the model. On the other hand, we might not be able to control over other aspects, such as the number of white blood cells present at the site of an infection. We refer to variables of this type as *state variables* because they help to describe the state of the system at any given time.

In this example, our only control variable is a binary decision whether or not to administer the drug on a given day, thus there are only two possible values for this control variable at each time step. For any given day, we may represent that the drug was administered that day by assigning that variable the value 1, and represent that the drug was not administered by assigning that variable the value 0. In this situation, then, a control schedule is a vector of length 365, of which each entry is either 0 or 1. Thus there are a total of  $2^{365}$  possible treatment schedules. However, it is likely that not all possible treatments will reduce the number of infected cells below the fixed level. The control schedules that *do* achieve this are *solutions* to the optimal control problem. The *solution space* is the set of all such solutions.

Recall that we are trying to find the best solution among many candidates. In order to do this, we must have some way of ranking individual solutions. We do this by introducing a *cost function*. We are attempting to steer the system to a particular state, and each solution achieves that goal at a certain cost. We wish to determine the best solution (i.e., the optimal control); thus we wish to find the solution that minimizes the cost function. Earlier we noted that in the immune system example the best solution was the solution that involved the least number of treatments; thus, in this example a reasonable cost function might involve the sum of the entries in the solution (i.e., the number of days on which the drug was administered). For the purposes of this chapter, the goal of the optimization and the optimal control problems will always be to **minimize** the associated cost function.

In our example, we wish to minimize  $T$ , the total number of days the drug is taken, and maximize  $I$ , the number of healthy cells. Then a reasonable cost function might be  $c(T, I) = T - I$ . Here we want to maximize  $I$ ; this is the same as minimizing  $-I$  (in this way we can always formulate our problem so that the goal is to minimize the cost function). However, it is perhaps more realistic that one of our optimization goals has a higher priority than the other. For example, we may suppose that maximizing the number of healthy cells is more important than minimizing the number of treatment days. In that case, we introduce weighting coefficients to alter the cost function. For example, it may become  $c(T, I) = T - 5I$ . This has the effect of increasing the importance of the healthy cell count when evaluating the cost of a control schedule. It is important, however, that we use the same cost function for each control schedule in order to maintain consistency.

In order to use our model to obtain results, we simulate the model a number of times and make observations on the results. We might, for example, administer a treatment every day. This particular solution is represented as a string of 365 consecutive 1's; we implement this in the agent-based model and count how many healthy cells there are after one year of simulated time. We then evaluate the cost function based on this information. Given that agent-based models are often stochastic in nature (meaning that while the rules are fixed, the

model dynamics depend on many random variables), results obtained from one simulation are typically not reliable. We eliminate such variation by simulating a given control schedule many times and using averages for evaluation in the cost function.

Of all the possible  $2^{365}$  solutions, there must be one that is better (or at least no worse) than every other solution. Strictly speaking, such a solution is referred to as an optimal solution. However, in many cases we cannot run all possible solutions because of the monumental amount of computing time involved. Thus, for the purposes of this chapter, we refer to the best solution *we are able to find* as the *optimal solution*, with the caveat that there may indeed be a better solution elsewhere in the solution space.

We now define the following terms, each of which have been illustrated in this example.

#### DEFINITION 12.2 (Terms).

- **Model parameters:** quantities that are part of the model specification. They have fixed values.
- **Model dynamics:** the interaction between the state variables in a model. In general, the model dynamics will be affected by the model parameters and the rules that govern the interaction of the variables.
- **State variable:** a variable whose value cannot be specified. State variable values affect the model dynamics; they are affected by the value of other state variables, model parameters, and control variable values.
- **Control variable:** variables whose value can be specified by the user. Altering the value of a control variable will (in general) have some effect on the resulting model dynamics.
- **Solution:** a sequence of inputs to the control variables. A full solution assigns a value to each control variable at each time step; a partial solution is a sequence wherein values are either only assigned to some of the control variables, or are assigned to the control variables at only certain time steps.

- **Solution space:** the set of all possible solutions. If  $p_1, p_2, \dots, p_n$  are the control variables and each parameter  $p_i$  (for  $1 \leq i \leq n$ ) can take on  $a_i$  possible values, and there are a total of  $t$  time steps, then the solution space will consist of

$$\prod_{i=1}^n a_i^t = a_1^t \cdot a_2^t \cdot \dots \cdot a_n^t = (a_1 a_2 \cdots a_n)^t$$

solutions (thus each solution is a vector of length  $t$ ).

- **Population:** a subset of the solution space. The population may be the entire solution space or a proper subset.
- **Cost function:** a function that assigns a value to each possible solution. In general, the cost function will depend upon a combination of state variables and other quantifiable aspects of the model dynamics.
- **Optimal solution:** In a general optimization problem the optimal solution is a solution that achieves the stated optimization goal in the best way. In an optimal control problem, it is the solution that is associated with the minimum value of the cost function.

Use the modified rabbit and grass model file `RabbitsGrass.nlogo` at <http://admg.vbi.vt.edu/software/netlogo-files/> to answer the questions in this section. Once you have opened the file, you will find a description of the interface settings under the **Info** tab.

EXERCISE 12.5. On the left-hand side of the model, you will notice sliders for the following values: `initial-rabbits`, `initial-grass`, `birth-threshold`, `grass-grow-rate`, `grass-energy`, `move-cost`, `world-size`, `poison-strength`. Are these model parameters or state variables? Explain.

EXERCISE 12.6. List three state variables and one control variable for this model.

EXERCISE 12.7. Suppose the control objective is to minimize the number of rabbits while also minimizing the number of days on which poison is used. What is the difference between a solution and the *optimal* solution?

EXERCISE 12.8. Click `restore-defaults`, set `poison-strength` to 0.5, and then click `setup`. Now, click `poison` repeatedly until you have run through the entire simulation time. Note the cost. Now click `degrade-poison` so that it is in the `On` position. Click `setup` and then repeatedly click `poison` again. What do you notice about the cost this time? Is it higher or lower? How can you explain the difference?

EXERCISE 12.9. Using the buttons `poison` and `don't poison` as you wish, run through the simulation several times, taking note of the cost for each solution. What is the lowest cost you are able to achieve? If you now reduce `poison-cost` to 5 and increase `rabbit-cost` to 15, you will notice that the cost when using the same schedule is now different. Is it lower or higher? Why do you think this is? Try to find a schedule that reduces the cost to less than 10,000. How about less than 5,000? Turn `degrade-poison?` to the `On` position and try to achieve similarly low costs. Do you notice any patterns in the solutions in each case?

EXERCISE 12.10. Run the same schedule twice (for 1 run each time). You should notice that the costs are different each time. Why does this happen? Explain why this is an important issue, and suggest a method for dealing with this issue.

EXERCISE 12.11. Click `restore-defaults`, then `setup`. Change `num-runs` to 10, then click `go`. Note the average cost over these 10 runs. Now change the number of runs to 20, then 40. What do you notice about the cost? Is it stabilizing? What is the least number of runs required in order to achieve a reliable cost? How would you justify your answer mathematically?

EXERCISE 12.12. What effect does altering `poison-cost` have on the cost of a fixed schedule? What effect does `rabbit-cost` have? What do you think are reasonable choices for these values if this were an actual field containing rabbits and grass? Justify your answer.

## 6. Scaling and aggregation

Our investigation of agent-based models and how to formulate control problems for them motivates this section and the next. Searching for an optimal solution in such models often

requires running many thousands of simulations, thus performing such simulations as quickly as possible is a primary concern. In this section, we discuss means of reducing the run-time and complexity of agent-based models via scaling and aggregation.

*Scaling* is a method of shrinking the size of an agent-based model in order to improve run time. In Exercise 12.1, we explored the “Rabbits Grass Weeds” model. The dimensions of this model are  $43 \times 45$  patches for a total of 1935 patches. The default number of rabbits is 150. In an attempt to scale this model, we may reduce the dimensions (and correspondingly, the initial number of rabbits). If we change the dimensions to  $25 \times 25$  for a total of 625 patches, we may choose the initial number of rabbits to be  $150 \times \frac{625}{1935} \approx 48$ . Our hope, then, is that the pertinent model dynamics remain the same at this reduced size, since in that case we can run all subsequent trials at this reduced size for a substantial decrease in run time.

The first question we need to answer is exactly what it means for the model dynamics to remain the same, and how we can verify that this is indeed the case. Since we are using the model with a specific control objective in mind, and the value of a solution relies on the associated cost function, this cost function will help us determine how to quantify whether our scaled model retains the pertinent dynamics. In Section 5, we used the “Rabbits and Grass” model to determine control schedules for poisoning the rabbits. In that case, our cost function relied on two parameters: the rabbit population, and the number of days in which poison was used. In particular, the day-by-day grass levels are not relevant to the cost function, nor are the energy levels of the rabbits; thus these parameters need not be preserved in the scaled model. In the scaled model, the number of days in which poison is used is unchanged, so this variable is also unaffected by scaling. Then the only parameter we need to preserve is the rabbit population. To be more specific, we need to answer the following question: what are the smallest dimensions we can use in the model (by scaling down the number of rabbits accordingly) so that the average day-by-day rabbit population dynamics in the scaled model follow the same pattern as those dynamics in the original model? We answer this question by simulating many runs at each size and keeping track of

the population levels for each. We can then calculate the correlation using any of a number of statistical methods; one such method is now presented.

**6.1. Correlating data sets.** Suppose we obtain two sets of data and wish to know how closely related they are. The absolute numbers may not tell the whole story, since the patterns in the data may remain similar even if the magnitude of the values change. Here we describe how to calculate *Pearson's sample correlation coefficient*, a real number  $r \in [-1, 1]$ , which estimates how closely correlated two data sets are. At  $r = 1$ , there is perfect correlation, such as between the ordered data sets  $\{1, 2, 3\}$  and  $\{2, 4, 6\}$ . At  $r = -1$ , there is perfect negative correlation, meaning that as the data increase in one set, they decrease by precisely the same proportional amount in the second data set (this value is obtained for data sets  $\{1, 2, 3\}$  and  $\{-8, -9, -10\}$ , for example). At  $r = 0$ , there is no connection between the two data sets at all (this value is obtained for data sets  $\{1, 2, 3\}$  and  $\{1, 2, 1\}$ ). Naturally, the larger the data sets the more informative the correlation coefficient. We now describe how to calculate  $r$  in general and then provide an example.

**DEFINITION 12.3** (Pearson's sample correlation coefficient). Let  $x, y$  be data sets consisting of  $n$  points (labeled sequentially as  $x_1, \dots, x_n$  and  $y_1, \dots, y_n$ ), and let  $\bar{x}$  and  $\bar{y}$  be the mean value of  $x$  and  $y$  respectively. Then Pearson's sample correlation coefficient  $r$  is defined as

$$r = \frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^n (x_i - \bar{x})^2 \sum_{i=1}^n (y_i - \bar{y})^2}}.$$

**Example.** Let  $x = \{150, 30, 40, 54, 72\}$  and  $y = \{72, 18, 10, 30, 40\}$ . Then  $\bar{x} = 69.2$  and  $\bar{y} = 34$ . Then

$$\begin{aligned}
 r &= \frac{\sum_{i=1}^5 (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^5 (x_i - \bar{x})^2 \sum_{i=1}^5 (y_i - \bar{y})^2}} \\
 &= \frac{(80.8 \cdot 38) + (-39.2 \cdot -16) + (-29.2 \cdot -24) + (-15.2 \cdot -4) + (2.8 \cdot 6)}{\sqrt{(80.8^2 + (-39.2)^2 + (-29.2)^2 + (-15.2)^2 + 2.8^2)(38^2 + (-16)^2 + (-24)^2 + (-4)^2 + 6^2)}} \\
 &= \frac{4476}{\sqrt{9156.8 \cdot 2328}} \approx 0.969.
 \end{aligned}$$

Thus we see that these two sets of data are in fact very closely positively correlated, even though the values in  $y$  are approximately one half of the corresponding values in  $x$ . Since agent-based models are generally stochastic in nature, the data obtained will seldom present a perfect description of the system, because an infinite number of simulations would be required. Thus sample correlation coefficients of 1 or  $-1$  are very highly unlikely. Then we may choose our desired correlation coefficient  $r$ , and when scaling the model we simply select the smallest dimensions that produce data whose correlation coefficient is at or above this level.

**6.2. Cost function analysis when scaling.** Once we have determined how small we can safely scale our model, we must also be cautious about numerical results obtained from this model, as indicated in the following example.

**Example.** Let  $r_i$  be the number of rabbits alive on day  $i$ , and let  $u_i$  be the control decision on day  $i$  (thus if we use poison then  $u_i = 1$  and if not,  $u_i = 0$ ). Let  $\vec{u} = [u_1 \ u_2 \ \dots \ u_n]$ , where  $n$  is the total number of simulated days. Suppose our cost function is  $c(\vec{u}) = a \cdot \sum_{i=1}^n r_i + b \cdot \sum_{i=1}^n u_i$ , where  $a, b \in \mathbb{R}$  are constants. Once we have scaled our model, we attempt to use it to determine the control schedule which minimizes this cost function. When doing so, we must

be careful to scale the constants  $a, b$  as well, since otherwise, we may obtain meaningless or misleading results, as the following results suggest.

**Example.** Let  $a = 100$ ,  $b = 2000$ ,  $n = 5$ ,  $\vec{u} = [0 \ 1 \ 1 \ 0 \ 1]$ ,  $\vec{v} = [1 \ 0 \ 0 \ 0 \ 0]$ . Suppose the average rabbit numbers for  $\vec{u}$  for the five simulated days are  $\{150, 30, 40, 54, 72\}$  and by scaling, we are able to reduce the model size and achieve average rabbit numbers  $\{75, 15, 20, 27, 36\}$ . Using another control schedule  $\vec{v}$  we obtain population levels  $\{150, 200, 40, 8, 10\}$  (and the corresponding scaled population values are  $\{75, 100, 20, 4, 5\}$ ).

Comparing the cost of the two schedules at the original size, we have:

$$c(\vec{u}) = 100(150 + 30 + 40 + 54 + 72) + 2000(0 + 1 + 1 + 0 + 1) = 40600,$$

$$c(\vec{v}) = 100(150 + 200 + 40 + 8 + 10) + 2000(1 + 0 + 0 + 0 + 0) = 42800.$$

**Without** scaling coefficients  $a, b$ , we would obtain the following costs using the scaled model:

$$c(\vec{u}) = 100(75 + 15 + 20 + 27 + 36) + 2000(0 + 1 + 1 + 0 + 1) = 23300,$$

$$c(\vec{v}) = 100(75 + 100 + 20 + 4 + 5) + 2000(1 + 0 + 0 + 0 + 0) = 22400.$$

From the original model, we conclude that  $\vec{u}$  is a better solution than  $\vec{v}$  (since the associated cost is lower), but from the reduced model we conclude that in fact  $\vec{v}$  is better than  $\vec{u}$ . This is due to the fact that when scaling the population values, even though the dynamics correlated perfectly, the coefficients now give less weight to the rabbit values, since these numbers are smaller in magnitude. In order to compensate for this, we must account for the rabbit numbers being halved by *doubling*  $a$ . At the same time, we need not scale  $b$  because the number of days remains constant at all model sizes. Scaling  $a$  and  $b$  accordingly gives the following revised costs for the scaled model:

$$c(\vec{u}) = 200(75 + 15 + 20 + 27 + 36) + 2000(0 + 1 + 1 + 0 + 1) = 40600,$$

$$c(\vec{v}) = 200(75 + 100 + 20 + 4 + 5) + 2000(1 + 0 + 0 + 0 + 0) = 42800.$$

Here we obtain essentially the same cost as the original model, and thus eliminate the discrepancy in cost.

The “Rabbits and Grass” model lends itself nicely to scaling because the initial distribution and location of rabbits and grass is random. Thus, when reducing the dimensions, we may still choose which patches have grass at random, and likewise may randomly choose where to place our rabbits at the start of the simulation. But what about models that are not so spatially homogeneous? Suppose the landscape included a river, or a rocky area in the upper right-hand corner of the map? In such cases, we cannot simply reduce dimensions because the spatial layout of the model may be critical to the dynamics. Thus in general, a more sophisticated approach is required for spatially heterogeneous models.

Suppose that in the “Rabbits and Grass” model, the field consists of a hill whose peak is at the center of the field. Going out from the peak of the hill, the altitude decreases; thus at the periphery of the map, the land is flat. Suppose further that we now distinguish between various levels of grass: each patch may have little or no grass, some grass, or a lot of grass. Finally, suppose the grass grows more abundantly at higher altitudes: thus at the peak of the hill there is a lot of grass, and at the periphery there is less. If we wish to model rabbits on such a landscape, it is important to maintain these characteristics as we scale the field.

The first step of our approach is to create a matrix that represents the physical landscape. We may do this by using the values 0, 1, 2, 3 (for example) to represent how much grass a given patch contains (with 3 being the most abundant and 0 representing little or no grass). Suppose our original model is  $10 \times 10$  and has layout as shown in Figure ??.

\*\* Insert Figure 12-original Here \*\*

In order to scale the model, we reduce the landscape using the **nearest neighbor** algorithm. First, we decide what dimensions we would like our reduced model to have; suppose it is  $n \times n$ . We then overlay an evenly-spaced grid of  $n \times n$  points over the original landscape (see Figure ??). Finally, we select values for each point by choosing the value of the neighbor nearest to that point (see Figure ??).

\*\* Insert Figure 12-overlay Here \*\*

\*\* Insert Figure 12-scaling Here \*\*

Note that this is only one algorithm that can be used for scaling a spatially heterogeneous model. Other methods include bilinear interpolation and bicubic interpolation; the reader is urged to explore the details of these algorithms on their own as this chapter provides only an introductory look at scaling.

*Aggregation* is another method of reducing computation and run time by simplification of the agent-based model. Rather than physically scaling the entire model, we may aggregate certain agents into groups and view each group as an agent. Thus there are fewer entities to keep track of, and fewer decisions that need to be made. This strategy is particularly helpful in models consisting of many agents of the same type, or many agents that follow the same set of rules. In modeling seasonal animal migration, for example, we may choose to aggregate a herd of antelope into one agent: the location of this agent would thus represent the average location of each individual in the herd. We can even represent certain antelope dying off and others being born by altering the size of the agent (e.g., as the antelope herd interacts with an aggregated prey agent such as cheetahs or lions, the size of each may expand or contract accordingly). Of course, such methods may not be possible depending on the aim of the model, and certain agents may not be amenable to aggregation. In particular, this strategy tends to be more difficult to implement in models with a high level of spatial heterogeneity or a high level of specification at the agent level.

Whereas scaling requires the dimensions of the model to be reduced (and other model parameters accordingly scaled), aggregation generally requires reconstruction of the model, since the scope of the model is altered as the agent structure is reformulated. In that sense, aggregation can be more involved than scaling. On the other hand, the long-term goal of both techniques is to improve run time and reduce computation, so the extra steps may be well worth the effort. A combination of both techniques can provide substantial decreases in run time and simplification without loss of pertinent model dynamics or detail.

The following exercises refer to the modified Rabbits and Grass model, `RabbitsGrass.nlogo`, available at <http://admg.vbi.vt.edu/software/rabbitsgrass-netlogo-files/>. Please read the **Info** tab to familiarize yourself with the details of this model.

EXERCISE 12.13. Run the model with the default values for `world-size` 44, 22, 5. Take a snapshot for each case, and examine the population plots. What differences do you notice? Do you think it is safe to run the model at size 22? How about at size 5? What is the smallest reliable size, based on the population graphs?

EXERCISE 12.14. Note in the cost function that we multiply by  $\frac{150}{\text{initial-rabbits}}$ . Why is this?

EXERCISE 12.15. Turn `scale-rabbits?` off and run the model several times at various sizes. What differences do you notice in the population graphs? What is the effect of this option? Are results at smaller sizes more or less reliable when `scale-rabbits?` is off?

EXERCISE 12.16. When `scaling-rabbits` is on and the world size is reduced, the rabbits become larger (graphically). Is reducing the number of rabbits an example of scaling or of aggregation? Justify your answer.

EXERCISE 12.17. What other methods besides those mentioned in this section could be used to improve the run time of this model (i.e., to make it more computationally efficient)?

## 7. A heuristic approach

In previous sections we have discussed optimal control as it relates to agent-based models, and techniques for improving the run time of such models. In theory, the process of finding the optimal solution requires only that we evaluate the entire solution space and choose the solution that minimizes the cost function. In practice, however, this remains unfeasible. For one thing, it is quite possible that the solution space is infinite; and even if it is finite, it is often the case that there are far too many solutions to possibly implement them all (in terms

of realistic run times), even in models that have been subjected to scaling and aggregation techniques.

For example, a discrete model of the immune system might involve many millions of cells each will have its own rules describing its interaction with other cells, and may also have many variables attributed to it. Analytic methods fail in these cases, and exhaustive enumeration by computer is unfeasible due to the limitations of computer processing speeds. In fact, as the processing power of computers grows, so too does the possibility for creating models that increase in complexity, so that it is impossible to be certain that computers will ever be able to ‘catch up’ to the complexity of the models.

For this reason, most of the optimization and optimal control methods currently used in discrete modeling (and with agent-based models in particular) are in the form of heuristic algorithms. An algorithm is a formal set of rules used in order to obtain a solution to a problem. A heuristic algorithm is a specific type of algorithm that conducts a ‘smart’ search of the solution space. It may make use of certain aspects of the problem to avoid having to search through every possible solution, or it may update its search method based on the input it receives from previously found solutions. These algorithms begin with a choice of control input values (i.e., full or partial solutions) and use various methods to refine the values so as to decrease the value of the associated cost function until no better solution can be found. While these methods do not guarantee an optimal solution, they do provide opportunity for analysis. In particular, if constructed and executed correctly, they are a vast improvement upon random searches of the solution space. One advantage to the heuristic approach is that it can be used with virtually any model. Since heuristic algorithms rely on results from simulation, it is not necessary to transform the model in any way in order to implement them. They provide a ‘brute force’ technique that, while not always optimal, can always be used when other methods may fail. In particular, heuristic algorithms provide a baseline for results obtained via other more rigorous methods. While a mathematician may be concerned with developing an algorithmic theory for explicitly finding a true optimal

solution, in practice scientists and other researchers are often satisfied with a solution that is better than any previously known. As such, the heuristic approach to optimal control is a valuable option.

Of course, every heuristic algorithm has associated risks and advantages, and the particular algorithm that is best suited for a given model or problem will depend on these. As an introduction to the heuristic approach, we describe a so-called *genetic algorithm* in some detail, in the context of the “Rabbits and Grass” model. We conclude this section with a list of other heuristic algorithms.

**7.1. Genetic algorithms.** Genetic algorithms (sometimes referred to as evolutionary algorithms) were originally developed as a means of studying the natural evolution process, and have been adapted for use in optimization since the 1960s [8, 12, 18]. Each solution is viewed as a chromosome. The chromosome is a string of values, or genes, each of which represents the value of one of the parameters. Each chromosome, then, represents an individual solution that can be implemented and whose associated cost can then be evaluated. A typical genetic algorithm functions in the following way: several chromosomes are selected to form a population. The cost of each is then evaluated. The chromosomes are ordered, beginning with the chromosome that produces the minimum cost. Half of the chromosomes are then selected to carry on to the next generation; that is, they will be kept in the list to be evaluated later. Then the process of breeding comes in to play: two random chromosomes from those that have been carried over are selected to serve as parents. A ‘child’ chromosome (i.e., a new solution) is then bred from these parents by some method of crossover (*uniform crossover* is a typical method in which there is an equally likely chance that the child will take each particular gene from either parent). The next step is *mutation*: each gene has the possibility of being mutated at random, changing from its current value to any admissible value of that gene. The child chromosome produced from such breeding is added to the next generation of solutions. This process is repeated until the remaining half of the new population has been re-populated with new child chromosomes. The entire process is then

repeated: the chromosomes are evaluated and the best are set aside, new chromosomes are bred from them, and so on.

The control objective with the “Rabbits and Grass” model was to determine a control schedule that minimized the number of rabbits while also minimizing the number of days on which poison was used. Suppose we wish to achieve this goal over the course of ten days. Then a solution is a vector of length 10, each entry of which is either 0 or 1. Suppose we begin with two randomly chosen solutions,  $p_1 = [0\ 1\ 0\ 1\ 0\ 1\ 0\ 1\ 0\ 1]$ ,  $p_2 = [1\ 1\ 1\ 0\ 1\ 1\ 0\ 0\ 1\ 0]$ . We create a ‘child’ solution by going through each gene (i.e., each entry in the parent solutions) and randomly selecting one of the values. See Figure ?? for an example of such a ‘child’ solution. The child solution is then subjected to mutation; see Figure ?? for an example.

\*\* Insert Figure 12-crossover Here \*\*

\*\* Insert Figure 12-mutation Here \*\*

The algorithm continues for a pre-determined number of steps, or until a certain condition is met. For example, we may choose to run the algorithm for 50 generations. Another method is to repeat the process until no better solution has been found for, say, ten consecutive generations. When the algorithm terminates, we choose the best current solution as our candidate for an optimal solution (note that there is no guarantee that the best solution found by the algorithm is actually the optimal solution; recall that we use *optimal solution* to mean the best solution we are able to find).

As with other heuristic algorithms, there are many variations and the one presented here is provided as a standard procedure. We may modify the algorithm, for example, so that the initial chromosomes are selected in a certain way: perhaps we wish to choose them at random, or perhaps we wish to choose chromosomes that are very different from one another (i.e., solutions that come from different regions of the solution space). We may modify the crossover process so that one parent is favored over another, or we may forego the mutation step altogether. The likelihood of mutation is another area where user input is important: a high level of mutation will result in more variation of child chromosomes, and thus will not

incorporate the relative fitness of the parent chromosomes as much. On the other hand, if the mutation step is not included then we run a greater risk of our solution converging to some solution that is only locally minimal; that is, it may be better than all of the solutions that are similar to it, but not necessarily better than solutions that are radically different (we may think of these solutions as being farther away in the solution space). The advantage of using a genetic algorithm is that there is inherently some level of stochasticity; that is, there is always the possibility of mutating to a better solution (as long as the mutation rate is nonzero).

For the following exercises, use Netlogo to open the file entitled **RabbitsGrass-GA.nlogo**, available at <http://admg.vbi.vt.edu/software/rabbitsgrass-netlogo-files/>. This file runs a genetic algorithm to attempt to determine the best poison schedule based on the cost function given in the model (the schedule which minimizes the number of rabbits and the days on which poison is used). Read the ‘Info’ tab for information on each of the options. In addition, you may need to examine the code in this model in order to complete all exercises.

EXERCISE 12.18. Explain the difference between roulette and tournament selection. How would roulette selection be different if we were interested in schedules with the *highest* cost?

EXERCISE 12.19. Explain the difference between uniform and 1-point crossover. Describe a control problem for which uniform crossover is more appropriate, and another for which  $n$ -point crossover is more appropriate.

EXERCISE 12.20. Explain the mutation methods **invert** and **neighbor swap**. Explain the advantages and disadvantages of each.

EXERCISE 12.21. Devise and explain methods for selection, crossover, and mutation other than those available in the model. Describe advantages of each.

EXERCISE 12.22. The **retention** slider determines how many of the top solutions should be carried over to the next generation. Describe a potential pitfall of setting this value to 0. Describe a potential pitfall of setting this value too high.

EXERCISE 12.23. Notice that the default number of runs is 100. What benefit is there to setting this value lower? What is the risk? What are the risks and benefits of setting this value higher?

EXERCISE 12.24. Run the algorithm at all default values, then run it again, only changing the poison so that it degrades. Take note of the best schedule found. What differences do you notice? Why do you think this is?

EXERCISE 12.25. Choose values for the various sliders and methods in an attempt to minimize the cost. Use your intuition as a starting point, then revise the values based on the output (make sure `write-to-file?` is turned on). What is the minimum cost you are able to achieve? What pattern (if any) do you notice in the best schedule found?

EXERCISE 12.26. Implement your ideas from Exercise 12.21 into the code and run the genetic algorithm using them. Do they perform better or worse than the original methods?

EXERCISE 12.27. Suppose the objective is simplified so that the goal is to simply minimize the number of rabbits, without regard to the poison cost. What do you think the best solution would be in this case? Alter the cost function in the code accordingly, and run the GA. Does the outcome reflect your intuition?

EXERCISE 12.28. Determine a new objective and alter the cost function accordingly. Run the GA. Discuss the pattern of the best schedule found, and state whether or not this seems to make sense in light of your cost function.

**7.2. Other heuristic algorithms.** Genetic algorithms represent only one class of heuristic algorithms. There are many others, and as mentioned earlier, each have their own advantages and disadvantages. The purpose of this section is to provide you with an overview of how heuristic algorithms can be used as a brute force approach to optimal control of agent-based models. Depending on the nature of the model and the control objective, the reader may also be interested in simulated annealing [17, 23], tabu search [5, 6, 7], ant

colony optimization [2, 3], and ‘squeaky wheel’ optimization [15, 22] (to name just a few). The combinations of these algorithms and the fine-tuning therein provide a large framework from which to apply heuristic algorithms to agent-based models, and new algorithms are constantly being developed. While more rigorous mathematical theory is currently being developed, this approach is sometimes the best available analytical tool.

## 8. Mathematical Framework for Representing Agent-based Models

As addressed earlier, many agent-based models are too complex to find global optimal control purely by simulation. One option one might pursue in this case is to translate agent-based models into a different type of mathematical object for which more mathematical tools are available. This is similar to the approach pioneered by Descartes and his introduction of a coordinate system. In the plane, for instance, a Cartesian coordinate system allows us to represent lines by linear equations, circles by quadratic equations, etc. If we now want to find out whether two lines in the plane intersect we can either carry out an inspection of the two lines as geometric objects or, alternatively, we can determine whether the resulting system of linear equations has a solution. Translating agent-based models into a formal mathematical framework then opens up the possibilities to use theory from mathematics to analyze them from a different angle rather than just with simulations. One framework that has been studied is that of so-called polynomial dynamical systems (PDS).

As in our running examples of rabbits and grass, agents in the models we consider here take on a finite number of states, such as a rabbit’s position on the grid or the amount of grass on a patch. This finite set of states can be viewed as the analog of the points that make up a line in the plane, to continue with the analogy of the Cartesian coordinate system. Applying the rules of the model in order to update the states of all the agents can be thought of as a function that maps the collection of discrete states to itself. Since the state sets do not carry any kind of structure we are simply dealing with a function from a finite set to itself, about which we can say little. However, let us now introduce the analog of a Cartesian coordinate system into this set in a way that provides extra tools for us, analogous to the

ability to use algebra to decide whether two lines in the plane intersect. We do this by way of an example.

Suppose that our set of states equals  $\{low, medium, high\}$ , which we can represent as  $\{0, 1, 2\}$ . We can easily turn this set into a number system by using so-called arithmetic modulo 3. That is, we can add and multiply the elements of this set according to rules that are identical to those used to multiply real numbers. Notice that the numbers in the set are all the remainders one can obtain under division by 3. Now we add and multiply “modulo 3,” that is,  $2 + 2 = 1$  and  $2 \cdot 2 = 1$  as well, that is the remainder of  $2 + 2 = 4$  under division by 3 is 1. The number system we obtain in this way is an example of a “finite field,” which we will use for the remainder of the chapter.

Given rules from an ABM that describe how the agents or variables change their states, we can find for every agent a polynomial that describes the given rule using element of the finite field such as  $\{0, 1, 2\}$  rather than  $\{low, medium, high\}$ . By constructing a polynomial for every single agent, we obtain a set of polynomials that describes the same behavior as the original ABM. The set is called a *polynomial dynamical system*.

**DEFINITION 12.4** (Polynomial dynamical system). Let  $k$  be a finite field and  $f_1, \dots, f_n \in k[x_1, \dots, x_n]$  be polynomials. Then  $F = (f_1, \dots, f_n) : k^n \rightarrow k^n$  is an  $n$ -dimensional polynomial dynamical system over  $k$  [25].

Here,  $k^n$  denotes the Cartesian product  $k \times k \times \dots \times k$  with  $n$  copies of  $k$ . For  $(a_1, \dots, a_n) \in k^n$  the function  $F$  is evaluated by

$$F(a_1, \dots, a_n) = (f_1(a_1, \dots, a_n), \dots, f_n(a_1, \dots, a_n)).$$

**Example.** Let  $k = \mathbb{F}_3 = \{0, 1, 2\}$  and  $F = (f_1, f_2, f_3)$ , where

$$f_1 = x_1 + x_2$$

$$f_2 = x_1x_2 + x_3 + 1$$

$$f_3 = x_1 + x_2 + x_3^2.$$

The PDS  $F$  maps points in  $\mathbb{F}_3^3$  to other points in  $\mathbb{F}_3^3$ , iteration of  $F$  results in a dynamical system.  $F$  has one fixed points, or steady state, namely  $(1, 0, 2)$  because  $F(1, 0, 2) = (f_1(1, 0, 2), f_2(1, 0, 2), f_3(1, 0, 2)) = (1, 0, 2)$ , note that all calculation are done “modulo 3” because  $F$  is over  $\mathbb{F}_3$ . Furthermore,  $F$  has a limit cycle or oscillation of length five: applying  $F$  repeatedly to  $(0, 2, 1)$  yields

$$(0, 2, 1) \rightarrow (2, 2, 0) \rightarrow (1, 2, 1) \rightarrow (0, 1, 1) \rightarrow (1, 2, 2) \rightarrow (0, 2, 1) \rightarrow (2, 2, 0) \rightarrow \dots$$

With software packages such as ADAM [11], one can compute the dynamics of a PDS and visualize it.

Most agent-based models can be translated into a polynomial dynamical system of this type. Each variable  $x_i$  represents an agent, a patch, or some other entity in the model. Each element in the finite field  $k$  represents a different state or condition of the variables. The corresponding polynomials  $f_i$  encode the behavior of the agent or patch as described in the model. In the rabbit and grass model, one could assign a variable  $x_i$  for each patch. The values of the finite field  $k = \{0, 1, 2, \dots, p-1\}$  then represent how many rabbits and how much grass is currently on the patch. The polynomials  $f_i$  determine the number of rabbits and amount of grass on patch  $i$  at the next iteration, given the current number of rabbits and grass.

**THEOREM 12.5 ([20]).** *Let  $f : k^r \rightarrow k$ ,  $k$  a finite field. Then there exists a unique polynomial  $g : k^r \rightarrow k$ , such that  $\forall x \in k^r, f(x) = g(x)$ .*

Any such mapping over a finite field can be described by a unique polynomial. Using Lagrange interpolation, we can easily determine the polynomial. Let  $f : k^r \rightarrow k$ . Then

$$g(x) = \sum_{(c_{i1}, \dots, c_{ir}) \in k^r} f(c_{i1}, \dots, c_{ir}) \prod_{j=1}^r (1 - (x_j - c_{ij})^{p-1}) \quad (12.1)$$

is the unique polynomial that defines the same mapping as  $f$ .

**Example.** Suppose  $k = \mathbb{F}_3$ ,  $r = 2$ , and the mapping  $f$  is defined on  $\mathbb{F}_3^2 = \{0, 1, 2\} \times \{0, 1, 2\}$  as follows:

$$f(0, 0) = 0$$

$$f(0, 1) = 1$$

$$f(0, 2) = 2$$

$$f(1, 0) = 1$$

$$f(1, 1) = 2$$

$$f(1, 2) = 0$$

$$f(2, 0) = 2$$

$$f(2, 1) = 0$$

$$f(2, 2) = 1$$

Then the polynomial  $g$  that defines the same mapping as  $f$  is constructed as following:

$$\begin{aligned}
g(x, y) &= 0 + \\
&\quad 1 \left( (1 - x^2)(1 - (y - 1)^2) \right) + \\
&\quad 2 \left( (1 - x^2)(1 - (y - 2)^2) \right) + \\
&\quad 1 \left( (1 - (x - 1)^2)(1 - y^2) \right) + \\
&\quad 2 \left( (1 - (x - 1)^2)(1 - (y - 1)^2) \right) + \\
&\quad 0 + \\
&\quad 2 \left( (1 - (x - 2)^2)(1 - y^2) \right) + \\
&\quad 0 + \\
&\quad 1 \left( (1 - (x - 2)^2)(1 - (y - 2)^2) \right) \\
&= x + y.
\end{aligned}$$

Converting an agent-based model into a polynomial dynamical system provides us with a conceptual advantage, rather than being limited to working with a computer simulation as our only means of analysis, methods and theory from abstract algebra and algebraic geometry can be used. For example, one might be interested in the steady states of a model, i.e., all the configurations of the system that do not change over time. Written as a polynomial dynamical system  $F : k^n \rightarrow k^n$ , these states are exactly the solutions to  $F(x) = x$ , the  $n$ -dimensional system of equations  $f_1(x) = x_1, \dots, f_n(x) = x_n$  [11]. Equivalently, the solutions are the points in the variety  $\mathcal{V}$  of the ideal generated by the polynomials  $f_1(x) - x_1, \dots, f_n(x) - x_n$ . For an introduction to varieties, we recommend [1].

When the polynomials describing the biological system have a special structure, other analysis methods are available. For example, for conjunctive networks, i.e., a PDS over  $\mathbb{F}_2$  where each polynomials is a monomial, the dynamics can be completely inferred by looking

at the dependency graph or wiring diagram of the PDS [14]. We can determine the fixed point and limit cycle structure of this PDS without using any simulation.

**8.1. Conway's Game of Life.** We have mentioned *Conway's Game of Life*, a cellular automaton as a special case of agent based models earlier in this chapter. Cells are agents that die or come to life based on a rule including their eight neighbors [4]. The Game of Life can be translated into a polynomial dynamical system. Each variable  $x_i$  represents a cell on the grid. Each polynomial  $f_i : \mathbb{F}_2^9 \rightarrow \mathbb{F}_2$  depends on  $x_i$ 's eight neighbors and itself and describes whether  $x_i$  will be dead (0) or alive (1) at the next iteration given the values of its neighboring cells. The fixed points of this system correspond to still lives such as blocks and beehives, 2 cycles to oscillators, e.g., blinkers and beacons. Working with the mathematical representation of the game, one can for example study still lives by using concepts from invariant theory using the symmetry of the rules as group actions.

It is straightforward to construct the polynomials using Lagrange's interpolation formula (12.1). The function  $g_i$  to interpolate describes the state of  $x_i$  based on its neighbors and itself according to the rules of over- and underpopulation. The polynomial  $f_i : \mathbb{F}_2^9 \rightarrow \mathbb{F}_2$  describing the behavior of cell  $x_1$  with neighbors  $x_2, \dots, x_9$  is shown here:

$$\begin{aligned} f_1 = & x_1x_2x_3x_4x_5x_6x_7x_8 + x_1x_2x_3x_4x_5x_6x_7x_9 + x_1x_2x_3x_4x_5x_6x_8x_9 + x_1x_2x_3x_4x_5x_7x_8x_9 + x_1x_2x_3x_4x_6x_7x_8x_9 + \\ & x_1x_2x_3x_5x_6x_7x_8x_9 + x_1x_2x_4x_5x_6x_7x_8x_9 + x_1x_3x_4x_5x_6x_7x_8x_9 + x_1x_2x_3x_4x_5x_6x_7 + x_1x_2x_3x_4x_5x_6x_8 + \\ & x_1x_2x_3x_4x_5x_7x_8 + x_1x_2x_3x_4x_6x_7x_8 + x_1x_2x_3x_5x_6x_7x_8 + x_1x_2x_4x_5x_6x_7x_8 + x_1x_3x_4x_5x_6x_7x_8 + \\ & x_2x_3x_4x_5x_6x_7x_8 + x_1x_2x_3x_4x_5x_6x_9 + x_1x_2x_3x_4x_5x_7x_9 + x_1x_2x_3x_4x_6x_7x_9 + x_1x_2x_3x_5x_6x_7x_9 + \\ & x_1x_2x_4x_5x_6x_7x_9 + x_1x_3x_4x_5x_6x_7x_9 + x_2x_3x_4x_5x_6x_7x_9 + x_1x_2x_3x_4x_5x_8x_9 + x_1x_2x_3x_4x_6x_8x_9 + \\ & x_1x_2x_3x_5x_6x_8x_9 + x_1x_2x_4x_5x_6x_8x_9 + x_1x_3x_4x_5x_6x_8x_9 + x_2x_3x_4x_5x_6x_8x_9 + x_1x_2x_3x_4x_7x_8x_9 + \\ & x_1x_2x_3x_5x_7x_8x_9 + x_1x_2x_4x_5x_7x_8x_9 + x_1x_3x_4x_5x_7x_8x_9 + x_2x_3x_4x_5x_7x_8x_9 + x_1x_2x_3x_6x_7x_8x_9 + \\ & x_1x_2x_4x_6x_7x_8x_9 + x_1x_3x_4x_6x_7x_8x_9 + x_2x_3x_4x_6x_7x_8x_9 + x_1x_2x_5x_6x_7x_8x_9 + x_1x_3x_5x_6x_7x_8x_9 + \\ & x_2x_3x_5x_6x_7x_8x_9 + x_1x_4x_5x_6x_7x_8x_9 + x_2x_4x_5x_6x_7x_8x_9 + x_3x_4x_5x_6x_7x_8x_9 + x_1x_2x_3x_4 + x_1x_2x_3x_5 + \\ & x_1x_2x_4x_5 + x_1x_3x_4x_5 + x_1x_2x_3x_6 + x_1x_2x_4x_6 + x_1x_3x_4x_6 + x_1x_2x_5x_6 + x_1x_3x_5x_6 + x_1x_4x_5x_6 + \\ & x_1x_2x_3x_7 + x_1x_2x_4x_7 + x_1x_3x_4x_7 + x_1x_2x_5x_7 + x_1x_3x_5x_7 + x_1x_4x_5x_7 + x_1x_2x_6x_7 + x_1x_3x_6x_7 + \end{aligned}$$

$$\begin{aligned}
& x_1x_4x_6x_7 + x_1x_5x_6x_7 + x_1x_2x_3x_8 + x_1x_2x_4x_8 + x_1x_3x_4x_8 + x_1x_2x_5x_8 + x_1x_3x_5x_8 + x_1x_4x_5x_8 + \\
& x_1x_2x_6x_8 + x_1x_3x_6x_8 + x_1x_4x_6x_8 + x_1x_5x_6x_8 + x_1x_2x_7x_8 + x_1x_3x_7x_8 + x_1x_4x_7x_8 + x_1x_5x_7x_8 + \\
& x_1x_6x_7x_8 + x_1x_2x_3x_9 + x_1x_2x_4x_9 + x_1x_3x_4x_9 + x_1x_2x_5x_9 + x_1x_3x_5x_9 + x_1x_4x_5x_9 + x_1x_2x_6x_9 + \\
& x_1x_3x_6x_9 + x_1x_4x_6x_9 + x_1x_5x_6x_9 + x_1x_2x_7x_9 + x_1x_3x_7x_9 + x_1x_4x_7x_9 + x_1x_5x_7x_9 + x_1x_6x_7x_9 + \\
& x_1x_2x_8x_9 + x_1x_3x_8x_9 + x_1x_4x_8x_9 + x_1x_5x_8x_9 + x_1x_6x_8x_9 + x_1x_7x_8x_9 + x_1x_2x_3 + x_1x_2x_4 + \\
& x_1x_3x_4 + x_2x_3x_4 + x_1x_2x_5 + x_1x_3x_5 + x_2x_3x_5 + x_1x_4x_5 + x_2x_4x_5 + x_3x_4x_5 + x_1x_2x_6 + x_1x_3x_6 + \\
& x_2x_3x_6 + x_1x_4x_6 + x_2x_4x_6 + x_3x_4x_6 + x_1x_5x_6 + x_2x_5x_6 + x_3x_5x_6 + x_4x_5x_6 + x_1x_2x_7 + x_1x_3x_7 + \\
& x_2x_3x_7 + x_1x_4x_7 + x_2x_4x_7 + x_3x_4x_7 + x_1x_5x_7 + x_2x_5x_7 + x_3x_5x_7 + x_4x_5x_7 + x_1x_6x_7 + x_2x_6x_7 + \\
& x_3x_6x_7 + x_4x_6x_7 + x_5x_6x_7 + x_1x_2x_8 + x_1x_3x_8 + x_2x_3x_8 + x_1x_4x_8 + x_2x_4x_8 + x_3x_4x_8 + x_1x_5x_8 + \\
& x_2x_5x_8 + x_3x_5x_8 + x_4x_5x_8 + x_1x_6x_8 + x_2x_6x_8 + x_3x_6x_8 + x_4x_6x_8 + x_5x_6x_8 + x_1x_7x_8 + x_2x_7x_8 + \\
& x_3x_7x_8 + x_4x_7x_8 + x_5x_7x_8 + x_6x_7x_8 + x_1x_2x_9 + x_1x_3x_9 + x_2x_3x_9 + x_1x_4x_9 + x_2x_4x_9 + x_3x_4x_9 + \\
& x_1x_5x_9 + x_2x_5x_9 + x_3x_5x_9 + x_4x_5x_9 + x_1x_6x_9 + x_2x_6x_9 + x_3x_6x_9 + x_4x_6x_9 + x_5x_6x_9 + x_1x_7x_9 + \\
& x_2x_7x_9 + x_3x_7x_9 + x_4x_7x_9 + x_5x_7x_9 + x_6x_7x_9 + x_1x_8x_9 + x_2x_8x_9 + x_3x_8x_9 + x_4x_8x_9 + x_5x_8x_9 + \\
& x_6x_8x_9 + x_7x_8x_9.
\end{aligned}$$

In the next section, we provide some polynomials for common rules in agent-based models. By providing such rules, we hope to simplify the process of translating an ABM to a PDS. The polynomials can be used as given or as a starting point to construct functions that represent more complex behavior.

## 9. Translating Agent-Based Models into Polynomial Dynamical Systems

*This section is authored jointly by Franziska Hinkelmann, Matt Oremland, Hussein Al-Asadi, Atsya Kumano, Laurel Ohm, Alice Toms, Reinhard Laubenbacher*

Discrete models, including agent-based models, are important tools for modeling biological systems, but model complexity may hinder complete analysis. Representation as a PDS provides a framework for efficient analysis using theory from abstract algebra. In this section, we provide polynomials that describe common agent interactions. In the previous section we describe how to interpolate agent behavior and to generate the appropriate polynomial.

However, for a variable with many different states, this method of interpolation results in long and complex polynomials that are difficult to expand, simplify, or alter. Thus we provide some general “shortcut rules” for constructing polynomials that describe key agent and patch interactions present in many ABMs. Each of the following polynomials exists in the finite field  $\mathbb{F}_p$ .

Since we are particularly interested in ABMs describing complex biological systems, we use the term concentration to describe the states of a patch variable (for example, concentration of white blood cells on a patch). In this chapter we describe several polynomials that describe both basic movement, and movement according to the state of the neighboring patches.

**9.1. Basic movement function.** We constructed various polynomials to describe the movement of an agent on an  $n$ -by- $n$  grid with torus topology, i.e., there are no boundaries to the grid, if an agent on the left most patch moves to the left, he appears on the right side of the grid, similarly for top and bottom. The where the  $x$  and  $y$ -coordinates of patches are numbered 0 to  $n - 1$ .

For an agent moving forward one patch per time step, we want an agent on patch  $x$  to move to patch  $x + 1$  unless the agent is on patch  $x = n - 1$ , in which case the agent will move to patch  $x = 0$  on the next step due to the torus topology of the grid, see Table 12.1. We constructed a similar polynomial for an agent moving backward one step per time

At time $t$	at time $(t + 1)$
patch 0	patch 1
patch 1	patch 2
patch 2	patch 3
$\vdots$	$\vdots$
patch $i$	patch $i + 1$
$\vdots$	$\vdots$
patch $n - 1$	patch 0

TABLE 12.1. Movement of an agent on a grid with  $n$  patches.

step, and using this polynomial along with the forward movement created a polynomial to

describe movement of one patch per time step in the  $x$ -direction when a vector  $u$  specifies direction each step (forward, backward, or no movement). We generalized these polynomials to describe movement of a fixed step length  $m$ .

- Agent moves forward one patch per time step:

$$f(x) = x + 1 + n \prod_{i=0, i \neq n-1}^{p-1} (i - x). \quad (12.2)$$

- Agent moves backward one patch per time step:

$$f(x) = x - 1 - n \prod_{i=1}^{p-1} (i - x). \quad (12.3)$$

- We can use these equations to construct a more general movement function  $f(x, u)$ , where  $u \in \{-1, 0, 1\}$  specifies the direction of the agent ( $u = -1$  backward 1,  $u = 0$  stay,  $u = 1$  forward 1). Note that  $a^{p-1} = 1$  for all  $a \neq 0 \in \mathbb{F}_p$ .

$$f(x, u) = x + u + un(u + 1)^{p-1} \prod_{i=0, i \neq n-1}^{p-1} (i - x) + un(u - 1)^{p-1} \prod_{i=1}^{p-1} (i - x).$$

- Agent moves forward  $m$  patches per time step:

$$f(x, m) = x + m + n \sum_{j=1}^m \left( \prod_{i=0, i \neq n-j}^{p-1} (i - x) \right).$$

- Agent moves backward  $m$  patches per time step:

$$f(x, m) = x - m - n \sum_{j=0}^{m-1} \left( \prod_{i=0, i \neq j}^{p-1} (i - x) \right).$$

PROOF. In order to show that equation (12.2) is the polynomial over  $\mathbb{F}_p$  that describes moving one space forward on an  $n$ -by- $n$  grid, we need to show that  $f(x) = x + 1$  for all  $x \neq n - 1$  and  $f(x) = 0$  for  $x = n - 1$ . Note that  $p = 0$  and  $(p - 1)! = -1$  in  $\mathbb{F}_p$ .

- Case:  $x \neq n - 1$ :  $f(x) = 1 + x + n(0) = 1 + x$ .
- Case:  $x = n - 1$ :

$$\begin{aligned}
f(n-1) &= 1 + n - 1 + n \prod_{i=0, i \neq n-1}^{p-1} i - n + 1 \\
&= n + n(1-n)(2-n) \cdots (n-2-n+1)(n-n+1)(n+1-n+1) \cdots (p-1-n+1) \\
&= n + n(p+1-n)(p+2-n) \cdots (p-1)(1)(2) \cdots (p-n) \\
&= n + n(1)(2) \cdots (p-n)(p+1-n) \cdots (p-1) \\
&= n + n(p-1)! = n + n(-1) = n - n \\
&= 0.
\end{aligned}$$

The proofs for the other movement functions are similar and left as an exercise to the reader.  $\square$

**9.2. Uphill and downhill movement.** In many applications, agents can scan their close environment and move towards a desired resource. Netlogo has this behavior implemented as “uphill” and “downhill”. “Uphill” moves an agent to the neighboring patch with the highest value of the desired variable. If no neighboring patch has a higher value than the current patch, the agent stays put. Since the variables are discrete, there may exist a tie between neighboring patches for highest (or lowest) concentration, in which case the agent moves towards the lowest arbitrarily numbered neighboring patch  $i$ . The polynomials describing the movement of agent  $x$  are the following.

- Uphill:

$$\begin{aligned}
f(x) &= \sum_{i=1}^8 \left( (1 - (C - I_i)^{p-1}) l_i \prod_{j=0}^{i-1} (C - I_j)^{p-1} \right) + (1 - (C - I_0)^{p-1}) \cdot l_0 + \\
&\quad \sum_{m=0}^{C-1} \left( \left( \prod_{k=m+1}^C \prod_{j=i}^8 (k - I_j) \right)^{p-1} \cdot \left( \sum_{i=1}^8 (1 - (m - I_i)^{p-1}) l_i \prod_{j=0}^{i-1} (m - I_j)^{p-1} + (1 - (m - I_0)^{p-1}) l_0 \right) \right)
\end{aligned} \tag{12.4}$$

• Downhill:

$$f(x) = \sum_{i=1}^8 \left( (1 - I_i^{p-1}) l_i \prod_{j=0}^{i-1} I_j^{p-1} \right) + (1 - I_0^{p-1}) \cdot l_0 + \sum_{m=1}^C \left( \left( \prod_{k=0}^{m-1} \prod_{j=i}^8 (k - I_i) \right)^{p-1} \cdot \left( \sum_{i=1}^8 (1 - (m - I_i)^{p-1}) l_i \prod_{j=0}^{i-1} (m - I_j)^{p-1} + (1 - (m - I_0)^{p-1}) l_0 \right) \right), \quad (12.5)$$

where the concentration ranges from 0 to  $C$ ,  $C$  represents the highest possible concentration,  $I_i$  is the concentration level at neighboring patch  $i$ , and  $l_i$  is the relative location of patch  $i$  from the current patch, see Table 12.2.  $I_0$  is the concentration of the current patch,  $I_1, \dots, I_8$  the concentrations of its 8 neighbors.

$l_8$	$l_1$	$l_2$
$l_7$	$x = l_0$	$l_3$
$l_6$	$l_5$	$l_4$

TABLE 12.2. Relative position of neighbors to agent  $x$ .

PROOF.

$$f(x) = \left( 1 - \prod_{i=0}^8 (C - I_i)^{p-1} \right) \left( \sum_{i=1}^8 \left( (1 - (C - I_i)^{p-1}) l_i \prod_{j=0}^{i-1} (C - I_j)^{p-1} \right) + (1 - (C - I_0)^{p-1}) l_0 \right) + \sum_{m=1}^{C-1} \left( \left( 1 - \prod_{i=0}^8 (m - I_i)^{p-1} \right) \prod_{k=m+1}^C \prod_{i=0}^8 (k - I_i)^{p-1} \right. \quad (12.6)$$

$$\left. \sum_{i=1}^8 (1 - (m - I_i)^{p-1}) l_i \prod_{j=0}^{i-1} (m - I_j)^{p-1} + (1 - (m - I_0)^{p-1}) l_0 \right).$$

It is straightforward to see, that (12.6) describes the movement to the neighboring patch with the highest concentration level:  $1 - \prod_{i=0}^8 (C - I_i)^{p-1}$  is 0 unless at least one of the  $I_i = C$ , i.e., one of the neighbors' concentrations level is the highest possible. In this case,  $f(l) = l + i$ . Indeed, the right hand of the first line evaluates to  $l_i$  for the  $i$  such that  $I_i = C$  ( $1 - (C - I_i)^{p-1} l_i$  is  $l_i$  if and only if  $I_i = C$ ). If there are several neighbors with concentration level  $C$ ,  $f(l)$  should evaluate to the neighbor with the smallest index. This is assured by multiplying with  $\prod_{j=0}^{i-1} (C - I_j)^{p-1}$ , which evaluates to 0, if an neighbor with a smaller index has concentration  $C$ . The second and third line in (12.6) are equivalent to the first row, as they describe movement to patches with concentration levels lower than  $C$ .  $\prod_{k=m+1}^C \prod_{i=0}^8 (k - I_i)^{p-1}$  assures that the second summand evaluates to 0 if a patch has a higher concentration than  $m$ . The proof for the down-hill movement (12.5) is similar and left as an exercise.  $\square$

EXERCISE 12.29. Based on the uphill and downhill polynomial, construct a polynomial  $f(l)$  that evaluates to the maximum concentration of its 8 neighbors and itself.

EXERCISE 12.30. Consider a 13 by 13 grid with torus topology. A rabbit moves two steps up and two to the right at every iteration. Construct a polynomial that describes a rabbit's movement. Use the polynomial to simulate the movement of a rabbit starting in the center of the grid.

EXERCISE 12.31. For the rabbit in Exercise 12.30, after how many iterations do you expect it to reach its starting position again? Confirm your answer by evaluating the polynomial.

EXERCISE 12.32. Construct the polynomial as in Exercise 12.30 for a grid of arbitrary size  $n$  by  $n$ .

EXERCISE 12.33. Consider a grid where each patch is covered with a *low*, *medium*, or *high* amount of grass. At each iteration, the rabbit moves to the neighboring patch with

the most grass, i.e., to one of the eight neighboring patches or it stays on the same patch. Construct a polynomial describing the rabbit's movement based on the amount of grass on the neighboring patches.

EXERCISE 12.34. Consider the grid from Exercise 12.33. Rabbits eat grass, and the amount of grass on a patch decreases by one level for every iteration that a rabbit occupies it, i.e., a patch with *high* grass changes to *medium* grass, if occupied by a rabbit, *medium* to *low*, and *low* remains *low*. Construct a polynomial that describes the amount of grass on a patch.

We provide these general polynomials and simplification techniques to aid in the transformation of an ABM into a PDS. Whereas large agent-based models may be too complex for efficient analysis, we hope that the algebraic structure of a polynomial dynamical system can be used to expedite computation of optimal control.

## 10. Summary

Agent-based models provide a very intuitive and convenient way to model a variety of phenomena in biology. The price we pay for these features is that the models are not explicitly mathematical, so that we lack mathematical tools for model analysis. For instance, many of these phenomena are connected to optimization and optimal control problems, as pointed out in this chapter, but no systematic methods are available for agent-based models to solve these. We have attempted here to do two things. Firstly, we described so-called heuristic local search methods, such as genetic algorithms, which can be applied directly to agent-based models. And we described a way in which one can translate an agent-based model into a mathematical object, in this case a polynomial dynamical system over a finite field. Many computational and theoretical tools are available for such systems. For instance, to compute the steady states of a polynomial system  $F = (f_1, \dots, f_n)$ , one can solve the system

of polynomial equations

$$f_1(x_1, \dots, x_n) = x_1, f_2(x_1, \dots, x_n) = x_2, \dots, f_n(x_1, \dots, x_n) = x_n.$$

There are several computer algebra systems available to solve such problems. To compute the steady states of an agent-based model, on the other hand, one is limited to extensive simulation, without guarantee of having found all possible steady states.

The chapter provides a snapshot of ongoing research in the field. The approach via polynomial dynamical systems, for instance, is very promising, but still lacks appropriate algorithms that scale to larger models. In addition to searching for such algorithms, further research in model reduction is taking place, as outlined earlier in the chapter. At the same time, other mathematical frameworks, such as ordinary or partial differential equations and Markov models are being explored for this purpose. Much work remains to be done but, in the end, a combination of better algorithms, improvements in hardware, and dimension reduction methods is likely to provide for us a tool kit that will allow the solution of realistic large scale optimization and optimal control problems in ecology, biomedicine, and other fields related to the life sciences.

**ACKNOWLEDGEMENTS.** M.O. and R.L. were partially supported by ARO Grant Nr. W911NF0910538. F.H. was partially supported by NSF Award 0635561.



## Bibliography

- [1] David A. Cox, John Little, and Donal O’Shea. *Ideals, Varieties, and Algorithms: An Introduction to Computational Algebraic Geometry and Commutative Algebra, 3/e (Undergraduate Texts in Mathematics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2007.
- [2] M. Dorigo, V. Maniezzo, and A. Coloni. Ant system: optimization by a colony of cooperating agents. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 26(1):29–41, feb 1996.
- [3] Marco Dorigo, Gianni Di Caro, and Luca M. Gambardella. Ant algorithms for discrete optimization. *Artificial Life*, 5:137–172, 1999.
- [4] M. Gardner. The fantastic combinations of John Conway’s new solitaire game “life”. *Scientific American*, 223:120–123, October 1970.
- [5] Fred Glover. Tabu search–part i. *Inform Journal on Computing*, 1(3):190–206, 1989.
- [6] Fred Glover. Tabu search–part ii. *Inform Journal on Computing*, 2(1):4–32, 1990.
- [7] Fred Glover. Tabu search: A tutorial. *Interfaces*, 20(4):74–94, 1990.
- [8] D.E. Goldberg. *Genetic algorithms in search, optimization, and machine learning*. Addison-Wesley, Reading, MA, 1989.
- [9] Volker Grimm, Uta Berger, Finn Bastiansen, Sigrunn Eliassen, Vincent Ginot, Jarl Giske, John Goss-Custard, Tamara Grand, Simone K. Heinz, Geir Huse, Andreas Huth, Jane U. Jepsen, Christian Jrgensen, Wolf M. Mooij, Birgit Mller, Guy Pe’er, Cyril Piou, Steven F. Railsback, Andrew M. Robbins, Martha M. Robbins, Eva Rossmannith, Nadja Rger, Espen Strand, Sami Souissi, Richard A. Stillman, Rune Vab, Ute Visser, and Donald L. DeAngelis. A standard protocol for describing individual-based and agent-based models. *Ecological Modelling*, 198(1-2):115 – 126, 2006.
- [10] F. Hinkelmann, D. Murrugarra, A.S. Jarrah, and R. Laubenbacher. A mathematical framework for agent based models of complex biological networks. *Bull Math Biol*, 2010.
- [11] Franziska Hinkelmann, Madison Brandon, Bonny Guang, Rustin McNeill, Grigoriy Blekherman, Alan V. Cuba, and Reinhard Laubenbacher. ADAM: Analysis of Discrete Models of Biological Systems Using Computer Algebra. *BMC Bioinformatics*, 12(1):295+, 2011.

- [12] John H. Holland. *Adaptation in natural and artificial systems*. University of Michigan Press, Ann Arbor, Mich., 1975. An introductory analysis with applications to biology, control, and artificial intelligence.
- [13] P.A. Iglesias and B.P. Ingalls, editors. *Control Theory and Systems Biology*. The MIT Press, Cambridge, MA, 2009.
- [14] Abdul Jarrah, Reinhard Laubenbacher, and Alan Veliz-Cuba. The dynamics of conjunctive and disjunctive boolean network models. *Bulletin of Mathematical Biology*, 72:1425–1447, 2010. 10.1007/s11538-010-9501-z.
- [15] David E. Joslin and David P. Clements. “Squeaky wheel” optimization. *J. Artificial Intelligence Res.*, 10:353–373 (electronic), 1999.
- [16] Donald E. Kirk. *Optimal Control Theory: An Introduction*. Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1970.
- [17] S. Kirkpatrick, C. D. Gelatt, Jr., and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.
- [18] John R. Koza. Evolution and co-evolution of computer programs to control independently-acting agents. In *Proceedings of the First International Conference on Simulation of Adaptive Behavior*, pages 366–375. MIT Press, 1991.
- [19] R. Laubenbacher, A. S. Jarrah, H. Mortveit, and S. S. Ravi. Mathematical formalism for agent-based modeling. In R.A. Meyers, editor, *Encyclopedia of Complexity and Systems Science*, pages 160–176. Springer, 2009.
- [20] Reinhard Laubenbacher and Brandilyn Stigler. A computational algebra approach to the reverse engineering of gene regulatory networks. *Journal of Theoretical Biology*, 229(4):523 – 537, 2004.
- [21] U. Ledzewicz, M. Naghnaeian, and H. Schattler. Optimal response to chemotherapy for a mathematical model of tumor-immune dynamics. *J Math Biol*, 64(3):557–577, Feb 2012.
- [22] Jingpeng Li, Andrew J. Parkes, and Edmund K. Burke. Evolutionary squeaky wheel optimization: A new analysis framework. *Evolutionary Computation*, Jan 2011. published online.
- [23] M. Pennisi, R. Catanuto, F. Pappalardo, and S. Motta. Optimal vaccination schedules using simulated annealing. *Bioinformatics*, 24:1740–1742, Aug 2008.
- [24] E.D. Sontag. *Mathematical Control Theory*. Springer Verlag, New York, NY, 2nd edition, 1998.
- [25] B. STIGLER, A. JARRAH, M. STILLMAN, and R. LAUBENBACHER. Reverse engineering of dynamic networks. *Annals of the New York Academy of Sciences*, 1115(1):168–177, 2007.
- [26] Alan Veliz-Cuba, Abdul Salam Jarrah, and Reinhard Laubenbacher. Polynomial algebra of discrete models in systems biology. *Bioinformatics*, 26(13):1637–1643, 2010.

- [27] J. Von Neumann. The general and logical theory of automata. In L.A. Jeffres, editor, *Cerebral Mechanisms of Behavior - The Hixon Symposium*, pages 1–31. J. Wiley & Sons, New York, 1951.
- [28] U. Wilensky. Netlogo. Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL, 2009. <http://ccl.northwestern.edu/netlogo/>.

12-original

0	0	0	0	0	0	0	0	0	0
0	0	1	1	1	1	1	1	0	0
0	1	1	1	2	2	1	1	1	0
0	1	1	2	3	3	2	1	1	0
0	1	2	3	3	3	3	2	1	0
0	1	2	3	3	3	3	2	1	0
0	1	1	2	3	3	2	2	1	0
0	1	1	1	2	2	2	1	1	0
0	0	1	1	1	1	1	1	0	0
0	0	0	0	0	0	0	0	0	0

FIGURE 12.1. Original  $10 \times 10$  grid representing topological landscape.

12-overlay

0	•	0	•	0	•	0	•	0	•	0
0	0	1	1	1	1	1	1	0	0	
0	•	1	•	1	•	2	•	1	•	0
0	1	1	2	3	3	2	1	1	0	
0	•	1	•	2	•	3	•	3	•	0
0	•	1	•	2	•	3	•	3	•	0
0	1	1	2	3	3	2	2	1	0	
0	•	1	•	1	•	2	•	2	•	0
0	0	1	1	1	1	1	1	0	0	
0	•	0	•	0	•	0	•	0	•	0

FIGURE 12.2. Original  $10 \times 10$  grid with  $6 \times 6$  overlaying points.

12-scaling

0	0	0	0	0	0
0	1	2	2	1	0
0	2	3	3	2	0
0	2	3	3	2	0
0	1	2	2	1	0
0	0	0	0	0	0

FIGURE 12.3. Resulting  $6 \times 6$  reduced grid.

12-crossover

$$\begin{aligned}
 p_1 &= [0101010101] \\
 p_2 &= [1110110010] \\
 p_{new} &= [0111110111]
 \end{aligned}$$

FIGURE 12.4. Uniform crossover to 'breed' a new solution.

12-mutation

$$[0111110111] \rightarrow [0011110110]$$

FIGURE 12.5. Values in red are subjected to mutation.